

WRDC-TR-90-8007
Volume VIII
Part 38

AD-A248 977



1

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 38 - Electronic Documentation System (EDS) Development
Specification

S. Barker, F. Glandorf

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

92-10047



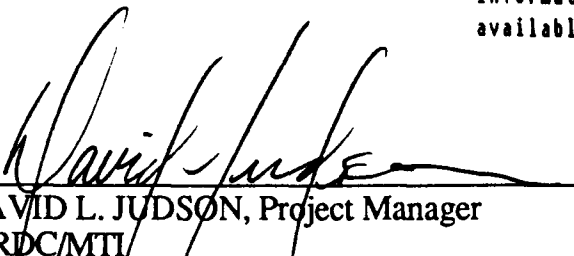
92 4 20 081

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

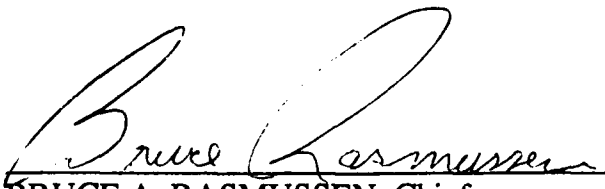
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations


DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:


BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DS 620344900		5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR- 90-8007 Vol. VIII, Part 38	
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services	6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI	
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209		7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF	8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464	
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533		10. SOURCE OF FUNDING NOS.	
1. TITLE Electroni See block 19 Specification		PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600
		TASK NO. F95600	WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Structural Dynamics Research Corporation: Barker, S., Glandorf, F.			
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4 / 1 / 87 - 12 / 31 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30	15. PAGE COUNT 61
16. SUPPLEMENTARY NOTES WRDC/MTI Project Priority 6203			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)	
FIELD	GROUP	SUB GR.	
1308	0905		
19. ABSTRACT (Continue on reverse if necessary and identify block number) This specification establishes the development, test, and performance requirements of an integrated set of computer programs know as the Electronic Documentation System. BLOCK 11: INTEGRATED INFORMATION SUPPORT SYSTEM Vol VIII -User Interface Subsystem Part 38 - Electronic Documentation System (EDS) Development Specification			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson		22b. TELEPHONE NO. (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC/MTI

EDITION OF 1 JAN 73 IS OBSOLETE

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Integration Technology Division, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W.A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. J. P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<u>SUBCONTRACTOR</u>	<u>ROLE</u>
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

	<u>Page</u>
SECTION 1.0 SCOPE	1-1
1.1 Identification.....	1-1
1.2 Functional Summary.....	1-1
SECTION 2.0 DOCUMENTS	2-1
2.1 Reference Documents.....	2-1
2.2 Terms and Abbreviations	2-1
SECTION 3.0 REQUIREMENTS	3-1
3.1 Computer Program Definition.....	3-1
3.1.1 System Capacities.....	3-1
3.1.2 Interface Requirements.....	3-1
3.1.2.1 Detailed Interface Definition.....	3-1
3.1.2.1.1 User Interaction.....	3-3
3.1.2.1.2 External File Processing.....	3-4
3.1.2.1.3 IISS Application Programs.....	3-4
3.1.2.1.4 Virtual Terminal.....	3-4
3.1.2.1.5 Neutral Data Manipulation	
Language - NDML.....	3-4
3.1.2.1.6 Standard Generalized Markup Language	
(ISO/8879)	3-4
3.1.2.1.7 Office Document Architecture/Office	
Equipment Interchange Format	
(ODA/ODIE - ISO 8613).....	3-5
3.1.2.1.8 Computer Graphics Metafile (CGM - ANSI	
X3.122).....	3-5
3.2 Detailed Functional Requirements.....	3-5
3.2.1 Document Processing Model.....	3-6
3.2.1.1 Editing Stage.....	3-6
3.2.1.2 Formatting Stage.....	3-7
3.2.1.3 Imaging Stage.....	3-7
3.2.2 EDS Functions.....	3-8
3.2.3 EDS Application Programs.....	3-9
3.2.3.1 Document Type Definition Builder.....	3-9
3.2.3.1.1 Inputs.....	3-10
3.2.3.1.2 Outputs.....	3-10
3.2.3.1.3 Processing	3-10
3.2.3.1.3.1 DTD Directory Form.....	3-10
3.2.3.1.3.2 Document Hierarchy Form	3-11
3.2.3.1.3.3 Document Hierarchy Menu Form.....	3-13
3.2.3.1.3.4 Document Hierarchy Subordinate Form...	3-14
3.2.3.1.3.5 Document Hierarchy Attribute Form...	3-15
3.2.3.1.3.6 Document Hierarchy Where Used Form .	3-16
3.2.3.1.3.7 Graph Capability in DTDBLD.....	3-16
3.2.3.2 SGML Tagger.....	3-17
3.2.3.2.1 Inputs.....	3-18
3.2.3.2.2 Outputs.....	3-18
3.2.3.2.3 Processing.....	3-19
3.2.3.2.3.1 Tagger Specification Screens.....	3-20
3.2.3.2.3.2 State Definitions.....	3-20
3.2.3.2.3.3 Character Class Definition Form.....	3-21

TABLE OF CONTENTS (Continued)

	<u>Page</u>
3.2.3.2.3.4 Pattern/State/Action Definition Form	3-22
3.2.3.2.3.5 C Definition Form.....	3-22
3.2.3.2.3.6 Pattern Specification Syntax.....	3-23
3.2.3.3 SGML Parser.....	3-25
3.2.3.3.1 Inputs	3-26
3.2.3.3.2 Outputs	3-26
3.2.3.3.3 Processing	3-26
3.2.3.4 SGML Context-Directed Editor (CDE).....	3-27
3.2.3.4.1 Inputs.....	3-27
3.2.3.4.2 Outputs	3-28
3.2.3.4.3 Processing	3-28
3.2.3.4.3.1 CDE Modes	3-28
3.2.3.4.3.2 CDE Document Editing	3-29
3.2.3.4.3.3 Logical Element Editing	3-32
3.2.3.4.3.4 Document Creation	3-32
3.2.3.4.3.5 Optional and Required Logical Elements.....	3-32
3.2.3.4.3.6 Using Entity Names in Documents.....	3-32
3.2.3.4.3.7 Assigning Attributes to Logical Elements.....	3-34
3.2.3.5 SGML Layout Macro Builder.....	3-34
3.2.3.5.1 Inputs.....	3-35
3.2.3.5.2 Outputs.....	3-35
3.2.3.5.3 Processing	3-35
3.2.3.5.3.1 DTD Selection Form	3-35
3.2.3.5.3.2 Document Profile Form	3-36
3.2.3.5.3.3 Macro Selection Form	3-37
3.2.3.5.3.4 Format Selection	3-39
3.2.3.6 SGML Formatter.....	3-41
3.2.3.6.1 Inputs	3-41
3.2.3.6.2 Outputs	3-41
3.2.3.6.3 Processing	3-41
3.2.3.7 Graphics File Translation	3-44
3.2.3.8 Text File Translation	3-44
3.3 Performance Requirements	3-44
3.3.1 Program Organization	3-44
SECTION 4.0 QUALITY ASSURANCE PROVISIONS.....	4-1
4.1 Introduction and Definitions	4-1
4.2 Computer Programming Test and Evaluation ...	4-1
SECTION 5.0 PREPARATION FOR DELIVERY	5-1

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-1	EDS Interface Diagram	3-2
3-2	EDS Application Interface	3-3
3-3	Document Processing Model	3-6
3-4	EDS Main Menu Screen	3-8
3-5	DTDBLD DTD Directory Form	3-10
3-6	Graph of Document Hierarchy for a Book	3-11
3-7	Sample Document Hierarchy for BOOK	3-12
3-8	DTD Document Hierarchy Menu Screen	3-14
3-9	Document Hierarchy Subordinate Form	3-14
3-10	DTDBLD Application Keypad	3-15
3-11	Document Hierarchy Attribute Definition Form	3-16
3-12	Tagger Main Menu	3-19
3-13	State Definition Form	3-20
3-14	Character Class Definition Form	3-21
3-15	Pattern/State/Action Definition Form	3-22
3-16	C Declaration Definition Form	3-23
3-17	CDE File Input Form	3-28
3-18	CDE Application Keypad	3-29
3-19	CDE Document Editing Screen	3-29
3-20	Display Tags Form	3-30
3-21	Display Entity Form	3-33
3-22	Display Attribute Form	3-34
3-23	Layout Editor - DTD Selection Screen	3-36
3-24	Document Profile Screen	3-37
3-25	Macro Selection Form	3-38
3-26	Tree Representation of DTD Segment	3-39
3-27	Format Selection Form	3-40
3-28	Formatter Input Form	3-42

SECTION 1

SCOPE

1.1 Identification

This specification establishes the development, test, and performance requirements of an integrated set of computer programs collectively known as the Electronic Documentation System, referred to as EDS.

1.2 Functional Summary

The Electronic Documentation System is intended to support the movement of a compound document through the document life cycle. Document life cycle phases are editing the document (creation/revision), formatting the document, imaging or printing the document, storing the document, and possible transfer of the document. The proposed system will include computer programs which will assist in these life cycle steps. Application programs and software tools defined below are designed to operate upon a document so as to prepare it to move or to actually move it from one phase in the document life cycle to another.

EDS is designed to process documents which have the following major characteristics:

- o They have a well-defined logical structure.
- o They can be composed of different content types (text, graphics, etc.).

A well-defined logical structure is a definition of which logical elements (paragraphs, sections, chapters) must be present in a document, and the order in which they must occur. A primary function of EDS is to provide mechanisms which enforce conformance to a predefined logical structure. This functionality enables EDS to support environments where documents must be written to specific standards, such as the ICAM standards. While EDS is designed to support the formatting, imaging, transfer, and storage of compound documents, it does not currently support compound document editing.

The operational philosophy of EDS is that the author of a document need not be concerned with the layout of the document. The primary concern for the author is document content and conformance to a document standard if one exists for the document being written. Once the logical structure and content are complete, the document can then be formatted and imaged using multiple presentation styles on multiple devices.

As previously mentioned, EDS is composed of an integrated set of software tools and application programs that operate upon a document during various stages in the document processing cycle. The major functions of EDS, along with the name of the programs which support these functions, are listed below. Detailed descriptions of each of these application programs (AP) are found in Section 3 of this document.

DTDBLD Create/revise document type definitions that define which logical elements may occur and the order in which they must occur within a document.

TAGGER Convert existing word processing documents to a "logical structure only" format.

PARSER Check specific documents for conformance to a documentation standard via the document type definition created by DTDBLD.

LMEDIT Create/revise formatting macros for a document class. Formatting macros contain instructions which define how the document will look (its presentation style) when the document is imaged.

DOCFMT Prepare a compound document for imaging. Text and graphics within the document are merged with document formatting commands.

CDE Guide a user through the creation of a document by using the document type definition as a model for what the logical structure of the document should contain.

VIRTRM Provide a virtual terminal which accepts neutral Virtual Terminal commands used to format the document.

DEV DVR Convert Virtual Terminal commands to device-specific commands.

GRATRA Provide graphics translation utilities from external graphics formats into Postscript EPS format.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SECTION 2

DOCUMENTS

2.1 References Documents

- [1] Systran, ICAM Documentation Standards, IDS150120000C, 15 September 1983.
- [2] International Organization for Standardization, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), ISO 8879, 15 October 1986.
- [3] International Organization for Standardization, Office Document Architecture/Office Document Interchange Format, ISO/DP 8613/1-6, October 1985 (Draft).
- [4] American National Standards Institute, American National Standard for Information Systems - Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information, ANSI X/3.122-1986, August 27, 1986.
- [5] Structural Dynamics Research Corporation, Form Processor User's Manual, UM 620344200A, 31 May 1988.
- [6] Structural Dynamics Research Corporation, Virtual Terminal Operator Guide, OM 620244000A, 31 May 1988.
- [7] M.E. Lesk, LEX - Lexical Analyzer Generator, IS Workbench for VAX/VMS Programmers Guide.
- [8] Structural Dynamics Research Corporation, Form Processor Development Specification, DS 620344700A, March 31, 1988.

2.2 Terms and Abbreviations

American Standard Code for Information Interchange (ASCII): The character set defined by ANSI x3.4 and used by most computer vendors.

Attribute: A characteristic used to qualify an element within a document.

Backus-Naur Form (BNF): A widely used notation form specifying the syntax of a language.

Character Set: A mapping of a character repertoire onto a code set such that each character is associated with its coded representation.

Common Data Model (CDM): Describes common data application process formats, form definitions, etc., of the IISS and includes conceptual, external, and internal schemas, and schema transformation operators.

Compound Document: A document which may contain mixed content (text, graphics, etc.).

Computer Graphics Metafile (CGM): A standard file format for the storage and retrieval of picture description information.

Computer Program Configuration Item (CPCI): An aggregation of computer programs or any of their discrete portions, which satisfies an end-use function.

Conforming SGML Application: An SGML application that requires documents to be conforming SGML documents, and whose documentation meets the requirements of this International Standard.

Conforming SGML Document: AN SGML document that complies with all provisions of this International Standard.

Context-Directed Editor: An EDS application which guides the user through the process of document creation and revision by using the document type definition as a model for which logical elements may be included in the document.

Cursor Position: The position of the cursor after any command is issued.

Descriptive Markup: Information added to a document that enables an application program to process the document.

Device Driver (DD): Software modules written to handle I/O for a specific kind of terminal. The modules map terminal-specific commands and data to a neutral format. Device Drivers are part of the UI Virtual Terminal.

Document Type Definition (DTD): Rules determined by an application that apply SGML to the markup of documents of a particular type. A document type definition includes a formal specification, expressed in a document type declaration, of the element types, element relationships and attributes, and references that can be represented by markup. It thereby defines the vocabulary of the markup for which SCML defines the syntax. A DTD can also include comments that describe the semantics of elements and attributes, and any application conventions.

Electronic Documentation System (EDS): An integrated set of software tools and application programs which operate upon a document through various stages of a document life cycle consisting of editing (creating/revising), formatting, imaging, storage, and transferring.

Element: A component of the hierarchical structure defined by a document type definition; it is identified in a document instance by descriptive markup, usually a start-tag and end-tag.

Element Declaration: A markup declaration that contains the formal specification of the part of an element type definition that deals with the content and markup minimization.

Entity: A collection of characters that can be referenced as a unit.

Field: Two-dimensional space on a terminal screen.

Form: A structured view which may be imposed on windows or other forms. A form is composed of fields. These fields may be defined as forms, items, or windows.

Form Definition (FD): Form definition Language after compilation. It is read at run-time by the Form Processor.

Form Definition Language (FDL): The language in which electronic forms are defined.

Form Editor (FE): A subset of the IISS User Interface that is used to create definitions of forms. The FE consists of the Forms Driven Form Editor and the Forms Language Compiler.

Form Hierarchy: A graphic representation of the way in which forms, items, and windows are related to their parent form.

Form Language Compiler (FLAN): A subset of the FE that consists of a batch process that accepts a series of form definition language statements and produces form definition files as output.

Form Processor (FP): A subset of the IISS User Interface that consists of a set of callable execution-time routines available to an application program for form processing.

Forms Driven Form Editor (FD FE): A subset of the FE which consists of a forms-driven application used to create Form Definition files interactively.

Imaging: The printing of a document.

IISS Function Screen: The first screen that is displayed after logon. It allows the user to specify the function to access and the device type and device name on which to work.

Integrated Information Support System (IISS): A test computing environment used to investigate, demonstrate, and test the concepts of information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Item: A non-decomposable area of a form in which hard-coded descriptive text may be placed and the only defined areas where user data may be input/output.

Layout Style: The specification of format and presentation for logical elements.

Layout Structure: The hierarchy of all layout elements (pages, frames, blocks, etc.) for a document.

Logical Structure: The hierarchy of all logical elements (paragraphs, sections, etc.) within a document.

Look-Ahead LR (LALR): An efficient, bottom-up parsing technique.

Message: Descriptive text which may be returned in the standard message line on the terminal screen. Messages are used to warn of errors or provide other user information.

Message Line: A line on the terminal screen that is used to display messages.

Neutral Data Definition Language (NDDL): A language used to manipulate and populate information in the Common Data Model (CDM) or IISS System Database.

Neutral Data Manipulation Language (NDML): A language developed by the IISS project to provide uniform access to common data, regardless of data base manager or distribution criteria. It provides distributed retrieval and single node update.

Office Document Architecture (ODA): A standard which supports the interchange of electronic compound documents in such a way as to allow their imaging, processing, or reformatting.

Office Document Interchange Format (ODIF): A standard for encoding document structures defined by the ODA which would enable the exchange of compound documents between systems operating within a MAP/TOP environment.

Operating System (OS): Software supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Page: Instance of forms in windows that are created whenever a form is added to a window.

Paging and Scrolling: A method which allows a form to contain more data than can be displayed at one time with provisions for viewing any portion of the data buffer.

Parser: An application program that determines how closely a document conforms to a document type definition which defines a specific documentation standard.

Physical Device: A hardware terminal.

Previous Cursor Position: The position of the cursor when the previous edit command was issued.

Qualified Name: The name of a form, item, or window preceded by the hierarchy path so that it is uniquely identified.

Standard Generalized Markup Language (SGML): A language for describing document structures, consisting of descriptive markup which is added to a document to indicate where logical elements such as sections and paragraphs begin and end.

Subform: A form that is used within another form.

Tag: Descriptive markup indicating the start or end of a logical element.

Tagger: An application program which provides a mechanism for automatically tagging existing documents which have been created by word processing systems.

User Data: Data which is either input by the user or output by the application programs to items.

User Interface (UI): IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: The User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Management System (UIMS): The run-time UI. It consists of the Form Processor, Virtual Terminal, Application Interface, the User Interface Services, and the Text Editor.

User Interface Services (UIS): A subset of the IISS User Interface that consists of a package of routines that aid users in controlling their environment. It includes message management, change password, and application definition services.

User Interface/Virtual Terminal Interface (UI/VTI):
Another name for the User Interface.

Virtual Terminal (VT): A subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

Virtual Terminal Interface (VTI): The callable interface to the VT.

Window: Dynamic area of a terminal screen on which predefined forms may be placed at run-time.

Window Manager: A facility which allows the following to be manipulated: size and location of windows, the device on which an application is running, the position of a form within a window. It is part of the Form Processor.

Yet Another Compiler Compiler (YACC): An LALR parser generator.

SECTION 3 REQUIREMENTS

3.1 Computer Program Definition

The Electronic Documentation System provides an integrated set of tools that enable the creation and revision of compound documents with a well-defined logical and layout structure.

3.1.1 System Capacities

EDS operates in the IISS environment on those hosts which support IISS.

It is capable of accepting input from and producing output for those devices which are supported within the IISS environment.

3.1.2 Interface Requirements

EDS interfaces with other IISS application programs by accepting graphical and textual input from these programs for inclusion into EDS documents. EDS includes application programs which translate external graphical and textual information into the standard file formats used by EDS. The file formats used by EDS will be based on current standards designated for document processing and exchange, as well as device-independent storage of graphics files.

EDS application programs which require user input use the User Interface Form Processor as a mechanism for accepting and validating user input.

3.1.2.1 Detailed Interface Definition

As shown in Figure 3-1, in order to process various text and graphic input/output from multiple external sources, EDS must provide interfaces to the external environment at multiple levels.

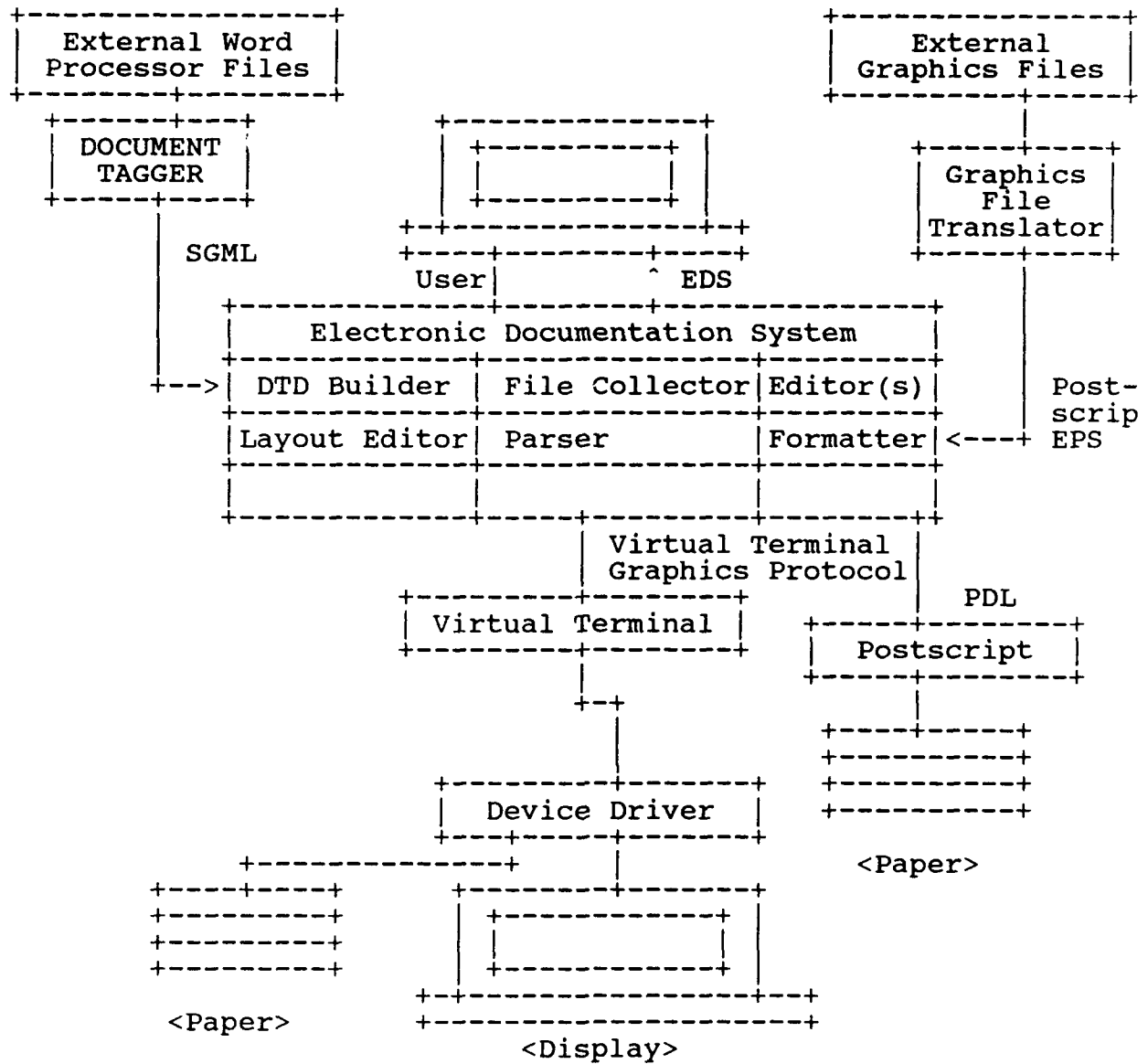


Figure 3-1 EDS Interface Diagram

These levels can be broken down into the following categories:

- o User interaction
- o External file processing
- o IISS application programs

The following sections provide descriptions of these interfaces.

3.1.2.1.1 User Interaction

All EDS application programs which require user input use the Form Processor. The Form Processor interface to an application consists of callable routines which are used to control user input/output from predefined forms into the application's data buffer. The Form Processor insures that all EDS application programs present a common forms-based user interface when user input is necessary.

Figure 3-2 illustrates how EDS applications interface with the Form Processor and Virtual Terminal components of the IISS environment.

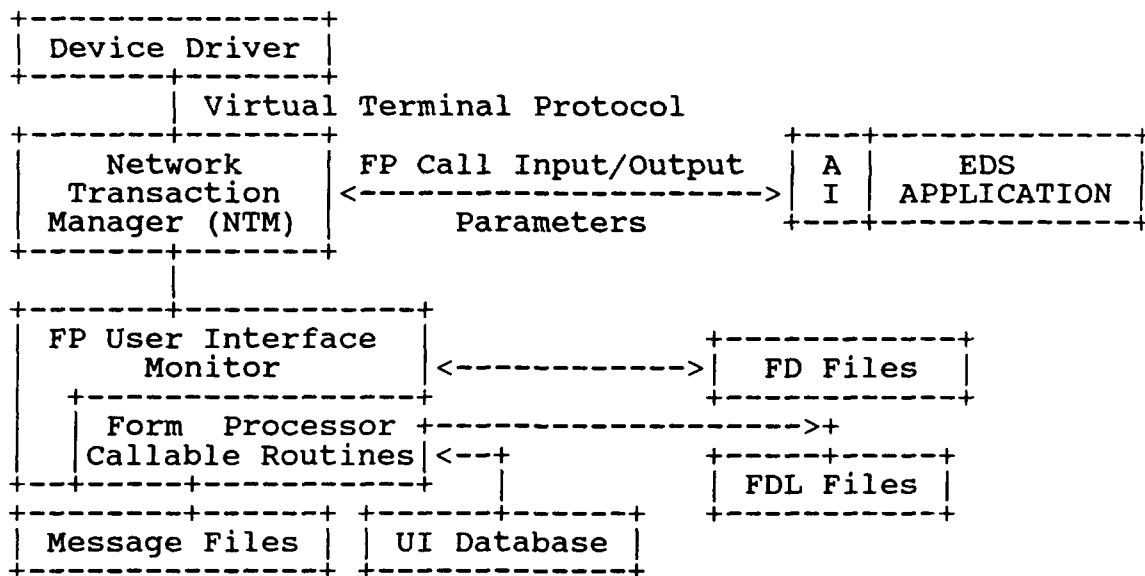


Figure 3-2 EDS Application Interface

3.1.2.1.2 External File Processing

External text and graphics files are converted by EDS into standard file formats which are based on current standards for document processing and exchange as well as device independent storage of graphics files. The standards which are used are SGML for text and Postscript for graphics. A brief description of these standards and how they are used within EDS is given in the next several sections.

3.1.2.1.3 IISS Application Programs

By using Postscript as a standard output format for graphics, IISS applications can produce graphics output for inclusion within an EDS document.

3.1.2.1.4 Virtual Terminal

The Virtual Terminal is a component of the Integrated Information Support System (IISS). It is a neutral terminal that presents a real terminal interface to the user. Although the Virtual Terminal looks like a real terminal, it is actually a neutral protocol based on a command set which is translated to and from device specific commands by device drivers. EDS uses the Virtual Terminal to image documents so that it may use all devices supported by the Virtual Terminal.

3.1.2.1.5 Neutral Data Manipulation Language - NDML

The EDS also provides support for accepting reports created by NDML statements. This would enable one to reference data base information which would be included within the document when it is formatted. This capability is important for those documents which must contain the most recent available data each time they are processed.

3.1.2.1.6 Standard Generalized Markup Language (ISO/8879)

SGML is a language for describing document structures. In an implementation of SGML, descriptive markup is contained within a document to indicate where logical elements such as sections and paragraphs begin and end. Descriptive markup is information added to a document that enables an application program to process the document. Descriptive markup is different from the procedural markup found in word processing systems or system utilities such as RUNOFF or TBL in the following ways:

- o SGML does not contain any information about the format of a document (bold, underline, etc.).
- o The tags which are used to define logical elements are human-readable and can be entered via text editors.

While SGML does not define a standard set of tags which are used for descriptive markup of a document, it does define a

language for describing a document structure. The set of SGML statements which define this structure is called a document type definition. A document type definition defines which logical elements may occur within the document and the order in which they may occur. Applications which are part of EDS use the document type definition as a source of information which can be used to check an instance of a document for conformance to a documentation standard.

SGML also provides for the inclusion of graphics files (of any format) within a document. Using the ENTITY statement, one can specify that the contents of an external file (either graphics or text) is to be included at a particular place in a document. SGML does not provide any facility for processing the graphics files. Other applications within EDS must interpret the graphical data and format both the text and graphics on a output device.

SGML was chosen for use in EDS because it specifies a method to both define and check the logical structure of a document.

3.1.2.1.7 Office Document Architecture/Office Document Interchange Format (ODA/ODIF - ISO 8613)

Office Document Architecture (ODA) is intended to support the interchange of electronic compound documents in such a way as to allow their imaging, processing or reformatting by the recipient.

ODIF is a method for the encoding of the document structure defined by ODA which would enable the exchange of compound documents between systems operating within a MAP/TOP environment.

In addition, as the ODA/ODIF standard develops, one can envision the use of ODA application programs within EDS to perform both logical and layout processing of compound documents.

3.1.2.1.8 Computer Graphics Metafile (CGM - ANSI X3.122)

The Computer Graphics Metafile provides a file format suitable for the storage and retrieval of picture description information. The file format consists of an ordered set of elements that can be used to describe pictures in a way that is compatible between systems of different architectures and devices of differing capabilities and design.

3.2 Detailed Functional Requirements

Before discussion of the functional requirements for each EDS AP, it is important for the user to understand how a document moves through EDS and where each application program resides in the document processing model.

3.2.1 Document Processing Model

The document processing model (Figure 3-3) illustrates that the functions Edit, Format, and Image are needed for the production of compound documents in EDS.

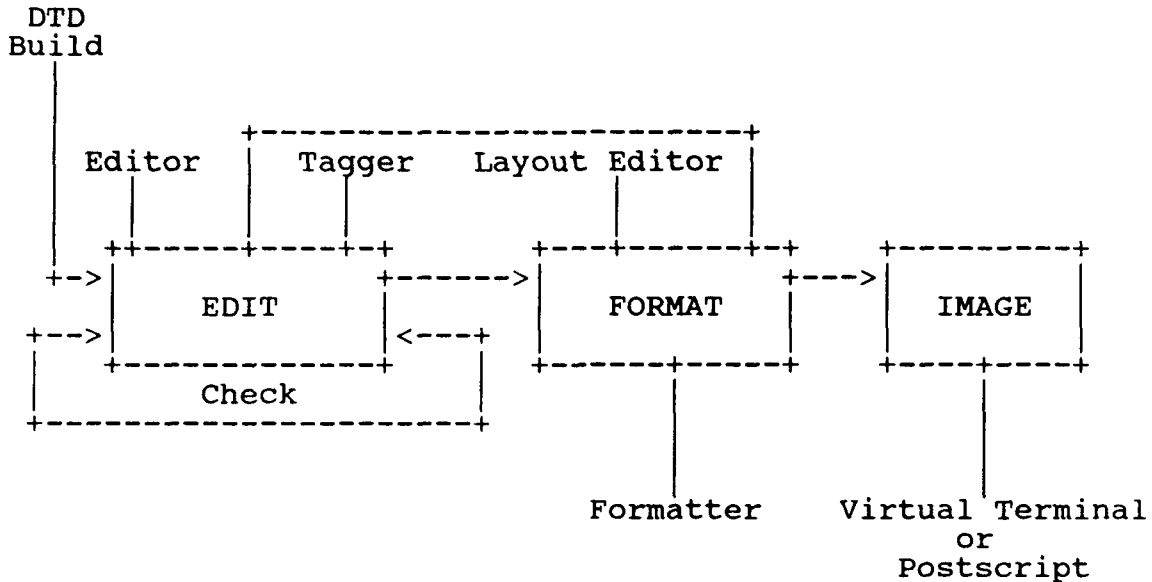


Figure 3-3 Document Processing Model

3.2.1.1 Editing Stage

The editing stage of the document processing model focuses on a document's logical structure, its content, and its conformance to the documentation standard to which it is being written. Because other documents which have been created by word processing systems contain formatting (layout) instructions within their files, the editing stage must also allow for the translation of these external documents into documents which contain only logical structure and document content. To do this, an EDS application program (TAGGER) first strips the document of all word processing codes. It then inserts the SGML tags that define the logical structure of a document into the document.

A document that contains SGML tags can then be edited, either by a standard text editor, where the author inserts the tags, or by a context-sensitive editor, where the system can either tag the document transparently or inform the author of which tags are possible at a current document context.

The SGML document type definition, which is created by the DTDBLD program, is what enables EDS to evaluate how closely a document conforms to some predefined documentation standard. The PARSER application compares the DTD to an instance of a

document and reports any errors found in the document markup. By not allowing any document to be passed to the formatting stage until it has been checked by the PARSER, EDS is able to insure that the logical structure requirements of a documentation standard have been met.

3.2.1.2 Formatting Stage

When the document content is complete, the layout (formatting instructions) structure must be added to the document to prepare it for imaging. At this time, the author's participation in creating the document ends, and responsibility for the document is transferred to someone in a document publishing role.

In addition to logical structure requirements, it is common that documentation standards call for specific layout structures to be used when a document is printed. For example, a documentation standard may require that all section titles be centered and underlined, or that all figure titles be printed in a certain font. In order to conform to these layout requirements, the Layout Editor application program enables one to specify how logical structure elements in a document look when the document is imaged. By building different document profiles, a document can be imaged with multiple layout styles.

Once a document profile is established for a given document class, the FORMATTER application program can format an instance of a document by replacing logical structure tags with formatting commands contained in the document profile. This formatted document can then be sent to the document imager (which in EDS is part of the Formatter) for translation of formatting commands into a Virtual Terminal command language or Postscript PDL.

The separation of logical from layout structure in environments where documents are large offers a number of advantages over document creation with word processing systems which embed formatting codes for layout structure. The main advantages are:

- o The author of a document can concentrate on the content and logical structure of a document.
- o Multiple authors of a document are assured that respective sections will have the same layout style.
- o Documents can be reformatted easily to meet new requirements when a layout requirement in a documentation standard changes.

3.2.1.3 Imaging Stage

The imaging process uses the layout information contained in the document to present a document on an output device. As mentioned above, this involves the translation of the formatting

commands contained in a document instance. The translation may be into the Virtual Terminal command language or into another language. The document could then be imaged on any device which is supported by the Virtual Terminal.

Consideration should be made for the ability of the translated language and supported devices to incorporate all aspects of the document instance. In some cases, a "what you see is what you get" (WYSIWYG) would not be able to incorporate all aspects of a document instance. The use of Postscript for imaging a document instance has proven effective and it is this language that supports EDS.

3.2.2 EDS Functions

When EDS is invoked, the user is presented with the Main Menu form shown in Figure 3-4.

```
+-----+
| IISS Electronic Documentation System   4/30/87 12:00:00 |
|                                     |
|           Document Processing Functions           |
|                                     |
|           E      Edit                      |
|           C      Check                     |
|           F      Format                     |
|           T      Translate                  |
|           P      Print                     |
|                                     |
|           Select Function >                |
|                                     |
+-----+
```

Figure 3-4 EDS Main Menu Screen

Access to EDS functions is controlled by the User Interface Services Application Definition service. As each function is selected, either a submenu is displayed or a separate application is started which initiates its own dialog with the user via the Form Processor.

The Edit function displays a menu which enables the user to use either the SGML Context-Directed editor or a standard text editor.

The Check function activates the SGML Parser application.

The Format function enables the user to format a document prior to printing. This function activates the Formatting application program.

The Translate function displays a menu of available translator application programs for both text and graphics.

The Print function permits the user to specify where a formatted document may be printed.

3.2.3 EDS Application Programs

The following sections provide a detailed description of each EDS application program.

3.2.3.1 Document Type Definition Builder

The Document Type Definition Builder (DTDBLD) is a forms-driven application which enables users who do not know the syntax of SGML but know (through document analysis) what the relationships between the elements of a document are, to build document type definitions (DTD). For example, in defining the logical structure of a book, one knows that a book contains chapters, chapters contain paragraphs, etc. The logical structure of a document can be expressed as a hierarchy consisting of parents, children, and groups of parents and children. The DTDBLD application program enables a user to define the hierarchy of a document by expressing the relationships between the parents and children of logical elements within a document. Information about the order of logical elements and how many times an element will occur (if at all) must be supplied by the user.

Along with the document hierarchy, the DTDBLD application program enables user definition of attributes for elements within a document. Attributes are used to further qualify a document element. For example, a book may have a status attribute which defines whether the book is a "final" or "draft" version.

As mentioned above, the DTDBLD program enables one to build a document type definition without having to know how to express a document hierarchy in SGML syntax. Users that have an advanced knowledge of SGML can use a standard text editor to insert some of the more advanced features of SGML into the DTD if necessary. The Element and Attribute statements which are used by the DTDBLD program are sufficient for building DTDs which define a documents logical structure. Other statements, such as Entity, Shortref, and Datatag are used mainly in increasing SGML processing efficiency or decreasing the amount of markup needed in a document. The DTDBLD program preserves all statements included by the user within a DTD.

3.2.3.1.1 Inputs

The inputs to DTDBLD are:

- o User input via form fields of SGML elements.
- o SGML elements to be read in from a DTD file.

3.2.3.1.2 Outputs

DTDBLD outputs an SGML document type definition file written to conform to the Standard Generalized Markup Language standard. The DTD file is used by several applications within EDS as a source of information defining what the set of valid elements in a given document is and in what order the elements may occur.

3.2.3.1.3 Processing

3.2.3.1.3.1 DTD Directory Form

When the DTDBLD program is executed, the form shown in Figure 3-5 is displayed.

```
+-----+
| EDS - Document Type Definition Builder  4/30/87 |
|                                             |
|      DTD Name _____  Action (C, D, E) |
|                                             |
| Action  DTD Name      Description          |
|  ---    ---          ---                  |
|  ---    REQDOC       ICAM Requirements Document |
|  ---    DSDOC        ICAM Development Specification |
|  ---    MEMODOC      General Purpose Memo      |
|                                             |
+-----+
```

Figure 3-5 DTDBLD DTD Directory Form

This form displays a scrollable list of all existing document type definitions. The scrolling of DTD directory entries is accomplished using the Scroll/Page mode defined in the IISS Terminal Operator Guide.

The user can perform three actions on any existing DTD: EDIT (E), COPY (C), and DELETE (D). The system will automatically search for and display any existing DTDs. Operations upon an existing DTD can be accomplished in two ways:

- o The action and the DTD name are entered on the first input line of the form, followed by the <ENTER> key.
- o The cursor is moved via the <TAB> key to the desired DTD and the command is entered into the action field, followed by the <ENTER> key.

To create a new DTD, the user must enter the DTD name and the Edit command. All Delete operations specified are revalidated by the DTDBLD program.

3.2.3.1.3.2 Document Hierarchy Form

A document type definition (DTD) is an SGML representation of the hierarchy of all logical elements within a document. As an example, Figure 3-6 is a partial graph of the document hierarchy for a document whose root element is BOOK.

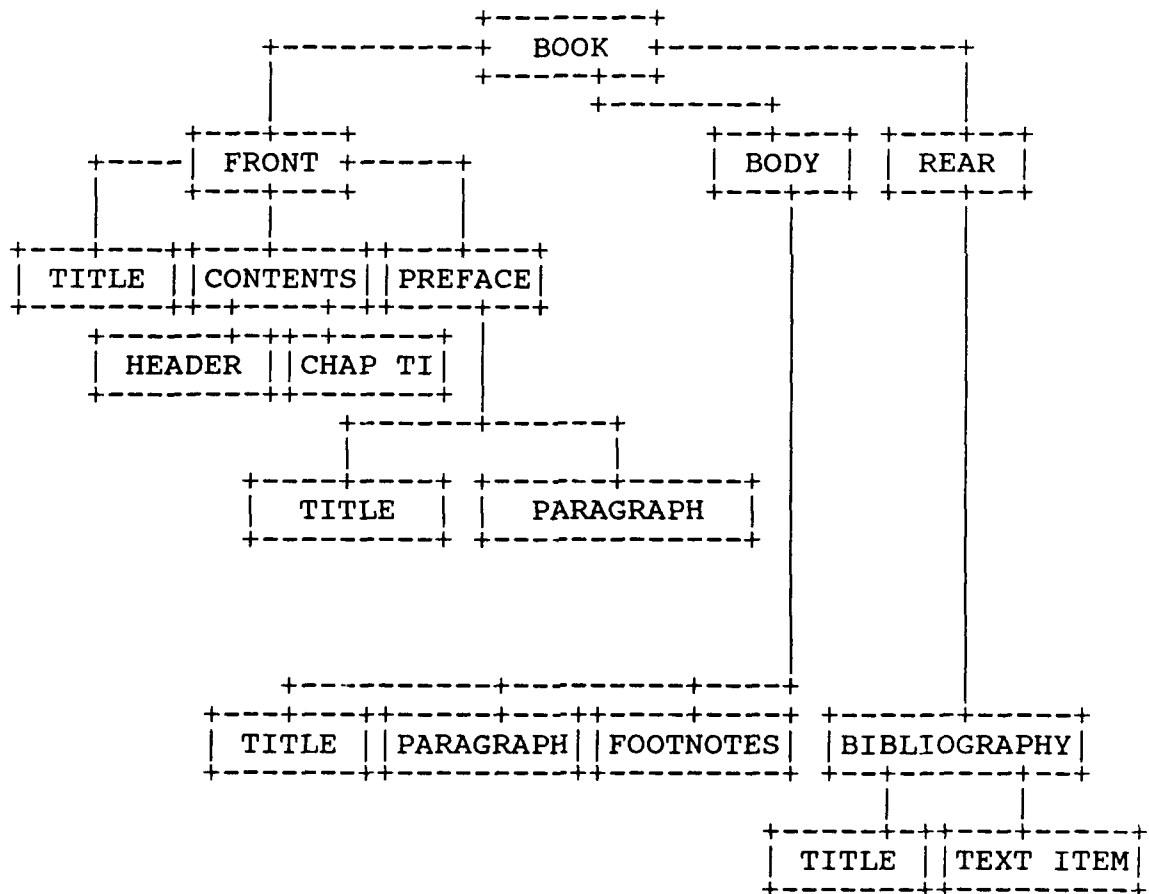


Figure 3-6 Graph of Document Hierarchy for a BOOK

Although the graph does not specify how many times each element may occur (if any), it does express parent-child relationships within the document hierarchy.

As mentioned above, the purpose of DTDBLD is to enable a user to define a document structure without having to express this structure in SGML syntax. This task is accomplished in DTDBLD by using the document hierarchy form as shown in Figure 3-7.

Document Hierarchy						
4/30/87 12:00						
Level	Common Name	Generic Identifier	Total Instance	Order	Group May Occur	Inst May Oc
0	BOOK	BK	1	SEQ	Z1	1
1	FRONT MATTER	FM	1	SEQ	Z1	1
2	TITLE	TI	1	SEQ	1M	1
2	CONTENTS	CN	1	SEQ	1M	
3	HEADER	HC	1	SEQ	1M	
3	CHAPTER TITL	T	1	SEQ	Z1	
2	PREFACE	PR	1	SEQ	Z1	
3	TITLE	TI	2	SEQ	1M	
4	PARAGRAPH	P	2	SEQ	1M	
1	BODY	BD	1	SEQ	Z1	
2	CHAPTER	CH	1	SEQ	1M	
3	TITLE	TI	3	SEQ	1M	
3	PARAGRAPH	P	1	SEQ	1M	

Figure 3-7 Sample Document Hierarchy for BOOK

Figure 3-7 contains entries for our example document hierarchy of BOOK.

When entering a document hierarchy, the user must enter the elements in their hierarchical order. This means that levels for each entry at the highest level must be fully specified before starting on the next element at the next highest level (equivalent to walking the tree in a depth first manner). By entering elements in this manner, the parent-children relationships of logical elements are defined.

In addition to the LEVEL and COMMON NAME for a document element, the following additional information must be specified in order to completely specify a documents logical structure.

Generic Identifier - This is shorthand notation for the Common Name and is to be used as the SGML tag name.

Order - Defines the order in which elements on the same hierarchical level must occur.
Possible values are:

SEQ - elements occur in required order
NSEQ- elements occur in any order
OR - one and only one element occurs
NONE- there are not chielements may occur only once

Occurrence - Defines how often an element may occur.
Possible values are:

Z1 - Zero or one time
ZM - Zero or many times
1M - One or many times

The first line in the Document Hierarchy form is used as a "search for" template. The user enters a value into one or more fields of the first line and the system searches for and displays the first element that matches the entered values.

Scrolling/Paging of document structure elements is supported via the Scroll/Page mode defined in the IISS Terminal Operator Guide.

3.2.3.1.3.3 Document Hierarchy Menu Form

The Document Hierarchy Menu form (Figure 3-8) is used to provide a menu of functions which enable a user to either display information about the document hierarchy or further qualify previously defined document elements.

Document Hierarchy Menu		4/30/87 12:00		
Subordinates	0			
Attributes	0	Common	Generic	Total
Where Used		Name	Identifier	
Graph		BOOK	BK	1

Figure 3-8 DTD Document Hierarchy Menu Screen

3.2.3.1.3.4 Document Hierarchy Subordinate Form

The Subordinate form (Figure 3-9) displays either a list of all children of the specified parent or a list of all siblings of the specified parent.

Document Hierarchy Menu		4/30/87 12:00		
Subordinates	3			
Attributes	1	Common	Generic	Total
Where Used		Name	Identifier	
Graph		BOOK	BK	1
Common Name	Generic Identifier	Total	Order	Occurrence
FRONT MATTER	FM	1	SEQ	Z1
BODY	BD	1	SEQ	Z1
REAR MATTER	RM	1	SEQ	Z1

Figure 3-9 Document Hierarchy Subordinate Form

Using the Subordinate function, the user can textually view the document hierarchy. The Application keys (Figure 3-10) enable the user to "walk-through" a document hierarchy using single keystrokes.

PF1	PF2	PF3	PF4
7	8	9	-
NEXT OCCUR	PREV OCCUR	6	,
DISPLAY CHILD	DISPLAY SIBLING	3	Enter
0		.	

Figure 3-10 DTDBLD Application Keypad

The Application Keypad for the Document Hierarchy Subordinate Form is defined as follows:

- o <DISPLAY CHILD> - displays all children of the specified parent.
- o <DISPLAY SIBLING> - displays the siblings (common parent) of the specified parent.

3.2.3.1.3.5 Document Hierarchy Attribute Form

The Attribute function is used to further qualify document elements through the use of attributes. The Attribute form (Figure 3-11) enables the user to specify multiple attributes for a given element as well as a default value for each attribute. (The "Status" attribute for the BOOK example is illustrated in Figure 3-11).

Document Hierarchy Menu		4/30/87 12:00	
Subordinates	3		
Attributes	1	Common	Generic Total
Where Used		Name	Identifier
Graph		BOOK	BK 1

Attribute	Default
STATUS	DRAFT
	Value
	DRAFT, FINAL

Figure 3-11 Document Hierarchy Attribute Definition Form

3.2.3.1.3.6 Document Hierarchy Where Used Form

The Where Used form enables the user to display where a particular logical element is used within the document hierarchy. Two keys, <NEXT OCCURENCE> and <PREVIOUS OCCURENCE> (see Figure 3-10), are used to list each occurrence of a particular logical element. The same form used in the Subordinate function is used for the Where Used function.

3.2.3.1.3.7 Graph Capability in DTDBLD

Additional planned capability for Graphics in the UI will allow the EDS to accomodate that capability. The DTDBLD component will be able to support graphical representations of the document hierarchy.

3.2.3.2 SGML Tagger

The SGML Tagger enables one to specify a translation between a document in word processing format to an SGML tagged document. A translation program must be built for each unique word processor and document type definition that will be used to process documents. A forms-driven interface is provided to assist in the specification of pattern matching type statements that define the mapping between word processor formatting codes and document content to SGML tags. These statements are used as input for a generated LEX program that performs the actual translation of the word processing document.

Typically, word processing files contain both logical and layout structure information. Formatting codes (character sequences) are used to represent logical (ex. paragraph) and layout (ex. page break) information within the document. In order to tag a document, certain formatting codes are used to generate SGML tags when they are encountered. For example, if all Section titles in a document were always centered and underlined, then a specification statement used by the Tagger might state that when the character sequence <center><underline> is recognized, the tag <STITLE> should be written to the output buffer. A <figure> code might also be used to generate a <FIGURE> marker.

Along with the formatting codes, the actual content of a document can be used to generate SGML tags. For example, a user can specify that any time the character sequence SECTION <number> is recognized, the </SECTNUM> is written to the output buffer. Specifications using a combination of content and formatting codes are also possible.

In addition to the forms-driven interface, the Tagger consists of the following parts: lexical writer, run-time processor, and specification storage.

Once all specification statements are entered and validated, the Tagger invokes the lexical writer to produce a valid LEX program that corresponds to the specification of the lexical components supplied by the user.

The run-time processing runs LEX to produce a C program from the specification file, compiles the result and links it with a main driver program. This main program can then be used to tag a word processing document.

After the document has been tagged, it is automatically sent to the Parser. The output of the Parser (the error report) provides an indication as to how well the document has been tagged. The Parser also inserts all end tags in the document. The Parser uses the DTD as a source of information to decide when end tags must be generated. Corrections to the markup in the document can then be made with either the SGML CDE editor or a text editor.

The efficiency of any word processor/SGML translator is a function of how many patterns can be defined that can generate SGML tags. Although it might not be possible to completely tag a document (so that it passes error free through the Parser), it is important that the Tagger be able to insert the most common tags within the document. These tags will typically be those tags that are end nodes (ex. #PCDATA) in a document hierarchy (such as a paragraph).

3.2.3.2.1 Inputs

The inputs to the Tagger specification forms are lexical statements of the following types:

- o User C declarations/Initial C statements
- o Character class definitions
- o State declarations
- o Pattern declarations
- o Action declarations

An explanation of the syntax and use of these statements is provided below.

The input to the Tagger LEX program is a document in word processing format.

3.2.3.2.2 Outputs

The output of the Tagger specification interface is a LEX program.

The output of the Tagger LEX program is a document in which all word processing format codes have been removed. The document will contain only the descriptive markup (SGML tags) for logical structure elements and the document content.

3.2.3.2.3 Processing

The forms-driven user interface allows the user to create/revise specification statements which will be used to generate a LEX program. When the user selects the Tagger from the EDS Main Menu, the following form (Figure 3-12) is displayed.

```
+-----+
|               EDS - SGML Auto Tagging Facility               4/30/87               |
|                                                                 |
|   Auto Tag Definition File:  ds.1                               |
|                                                                 |
| <enter> - Save information to Auto Tag definition file          |
| <pf5>   - State definition                                       |
| <pf6>   - Character class definition                             |
| <pf7>   - Pattern/Action/State definition                       |
| <pf8>   - C declarations                                         |
|                                                                 |
+-----+
```

Figure 3-12 Tagger Main Menu

This form establishes the name of the load/save file and serves as an entry to the Tagger specification screens. By entering a file name and pressing <LOAD> (PF16), the previous work is cleared and the file is loaded. If a new specification is being started the template file for the appropriate word processor must be loaded. When <ENTER> is pressed, the file is saved. When one of the Action keys <PF5 - PF8> is pressed, a specification screen is displayed.

3.2.3.2.3.1 Tagger Specification Screens

Each one of the specification screens operate in the same manner. Each screen has a field at the top for entering new data, a field at the bottom of each group to indicate the end of the group, and a repeating field in the middle to display previously entered data. Placing the cursor on a field and pressing an <ACTION> key (PF5 - PF8) allows the user to modify the field by bringing up a detail screen. The content of the bottom field is not part of the specification and cannot be edited. Pressing <ENTER> adds the contents of the new data field to the group. Fields can be deleted by moving the cursor to the desired field and pressing <DELETE>.

Syntax checking is provided for each specification screen in order to shorten the amount of time needed to create a valid LEX program.

3.2.3.2.3.2 State Definitions

The state definition screen (Figure 3-13) contains a list of states which may be used in pattern/state and action definitions.

```

+-----+
|               EDS - SGML Auto Tagging Facility               |
|               State Definition                                |
|               State                                         Comment                                |
| +-----+ +-----+                                         |
| +-----+ +-----+                                         |
| TITLE                                           |
| TITLE1                                          |
| TOC                                             |
| BODY                                           |
| **END**                                        |
+-----+

```

Figure 3-13 State Definition Form

States are used to signify that 1) a certain event has occurred, and 2) specifications can be created which allow only certain actions to occur in certain states. For example, when a text SECTION <number> is recognized, the state SECTION will be set "on". A specification could then be defined which would recognize Section Titles (defined by <center><underline> formatting codes) only when the state SECTION was set on. States are identifiers and may appear in any order. If a state is used in the pattern/state and action definitions without being defined in the State Definition Form, a warning is issued. A comment may also be provided for each state.

3.2.3.2.3.3 Character Class Definition Form

The character class definition form (Figure 3-14) contains the list of character class names and their regular expression definitions.

EDS - SGML Auto Tagging Facility 4/30/87	
Character Class Definition	
Character Class	Regular Expression Definition
bold_on	"{#"
bold_off	"{\"

Figure 3-14 Character Class Definition Form

The purpose of the character class name is to serve as a mnemonic for a word processing formatting code. These class names can then be used in pattern definitions in place of a regular expression. All character class names must be defined via this form before they are used in a pattern. If not, a warning will be issued.

3.2.3.2.3.4 Pattern/State/Action Definition Form

The pattern/state/action definition form (Figure 3-15) contains a list of patterns and actions.

EDS - SGML Auto Tagging Facility Pattern/Action/State Definition	
Pattern	Action
{und_on}?"SECTION"{space}*{n /* Section */ **end**	

Figure 3-15 Pattern/State/Action Definition Form

The syntax of patterns is provided in the next section. An action can be any valid C statement that would normally occur in the body of a procedure. The special action "ECHO;" causes the contents of the input buffer to be output without formatting information. The special action "BEGIN state_name;" sets the current state to the specified state name. The current state name is reset if the state name is "0".

3.2.3.2.3.5 C Definition Form

The C Definition form (Figure 3-16) is used to insert C statements for the LEX module yylex. C statements can be used to define variables referenced in actions and to perform initialization of the output document file. For example, the user might wish to output several tags before processing of the word processing file begins.

EDS - SGML Auto Tagging Facility Pattern/Action/State Definition	
Pattern	Action
{und_on}?"SECTION"{space}*(n /* Section */ **end**	

Figure 3-16 C Declaration Definition Form

3.2.3.2.3.6 Pattern Specification Syntax

Patterns represent elements of a document that are to be tagged. These patterns consist of strings and operators. A string is a sequence of printable characters. An operator is one of the following characters:

" \ [] ^ - ? . * + | () \$ / { } % < >

The characters <space>, <> (angle brackets), ^ (karat), % (percent), and \$ (dollar sign) are special characters to LEX and should be enclosed in double quotes or preceded by a backslash. Warnings will be issued for improper use.

A pair of double quotes is used to enclose a literal string: "SECTION 5"

The backslash character (\) when prefixed to an operator is used to remove the special meaning and allow the operator to be used as an ordinary character: "This , \", is a double quote"

The karat may be used as the first character in [] so the Tagger recognizes any character NOT within the class. The following example recognizes any characters except a, b, or c: [^abc]

The dash (-) may be used in [] to indicate a range of characters to be recognized. The following example recognizes any character from a to z: [a-z]

The question mark (?) is used to indicate that the preceding element is optional. The following example recognizes the input strings "ac" and "abc": ab?c?

The plus sign (+) indicates that the preceding element may occur one or more times. The following example recognizes the input strings "abc", "abbc", and "abbbc": ab+c

The asterisk character (*) indicates that the preceding element may occur zero or more times. The following example recognizes the input strings "ac", "abc", or "abbc": ab*c

The bar character (|) indicates that the preceding or following element may be input. The following example recognizes the input strings "a" or "b": a|b

Parentheses are used to group elements so that they are considered a single element. The following example recognizes the input strings "abc" or "def": (abc)|(def)

The slash operator indicates that the preceding regular expression is recognized only if followed by the regular expression after the operator. The following example recognizes the input string "abc" if followed by "def": (abc/def)

Curly braces ({}) enclosing a name indicate that the name within the braces is a character class and should be expanded.

If the curly braces enclose a number, it indicates that the preceding element may occur the specified number of times. The following example recognizes the input strings "aa", "aaa", and "aaaa": a{2,4}

3.2.3.3 SGML Parser

The SGML Parser is the application program that determines how well an instance of a document conforms to a document type definition which defines a specific documentation standard. The document type definition is used to specify which logical structure elements (delimited by SGML tags) to expect within a document and how often these elements occur.

The Parser must perform a number of tasks when processing a document. The major tasks it must perform include:

- o Read in and validate the specified document type definition.
- o Resolve Entity References.
- o Scan the document to identify Tags, Entity References, and Data Strings.
- o Interpret the descriptive markup tags and attributes and check element relationships against the DTD.
- o Expand markup minimization.
- o Report errors found in the document markup.

The Parser conforms to the ISO 8879 standard for SGML. Initial efforts in the development of the Parser will concentrate on the implementation of the core features of the SGML standard. These features are as follows:

- o DTD validation of ELEMENTS, ATTRIBUTES, ENTITIES
- o Omittag start and end tag generation
- o Support for SGML declaration
- o Error generation for document markup
- o Resolution of ENTITY references (for SGML and markup)

3.2.3.3.1 Inputs

The input to the SGML Parser is an SGML Document file. This file may contain an SGML Declaration; if not, a system declaration will be input. The file must contain a document type definition either explicitly or by reference to another file.

3.2.3.3.2 Outputs

The output of the Parser is a fully marked-up document. A fully marked-up document consists of the full expansion of SGML tags (no minimization within the document and all entity references resolved - both string substitutions and external files. Any external references to graphics files are not resolved by the Parser. The Entity statement will be left in the fully marked-up document to indicate to other EDS application programs where graphics files are to be included within the document.

An error report is generated by the Parser if errors are detected in either the markup or the document type definition. The types of errors that the SGML Parser recognizes include the following:

- o An error in the SGML Declaration
- o An error in the Document Type Definition
- o Errors in the descriptive markup
- o Failure to resolve an entity reference

3.2.3.3.3 Processing

The SGML Parser conforms to a layered architecture. From lowest to highest the layers are: Manager, Lexical Analyzer, Parser, Document Follower, and the Output Writer.

The Input Manager is responsible for keeping track of the current source of input - the main document file, a referenced file, or an Entity definition. It does this by keeping a stack of input sources so that when a source is exhausted, an entity end signal is generated and input reverts back to the prior source. The Input Manager is also responsible for allowing any back-tracking required by the Lexical Analyzer.

The Lexical Analyzer is responsible for classifying the input characters and separating them into recognizable groups such as delimiters, keywords, numbers, etc. This separation is highly context-dependant; delimiters are recognized only in certain contextual modes, keywords are recognized within declarations but not within literals, etc. Most of the contextual information is generated and maintained within the Lexical Analyzer, but some may need to be generated by the Parser or Document Follower.

The Parser recognizes the various declarations and markup which may appear in an SGML document. Declaration information is recorded and used by the Input Manager, Lexical Analyzer, and Document Follower to affect their subsequent behavior. Any violations of the SGML syntax are reported by this layer.

The Document Follower is responsible for relating the marked-up document to the document type definition. It recognizes and interprets minimized and omitted markup. Any violations of the structure specified by the document type definition are reported by this layer.

The Output Writer is responsible for writing out the fully marked-up document. If it is determined that multiple output formats are desired for various applications, flags are provided to specify the desired format.

The SGML Parser is a batch application which does not interact with the user. The interface is similar to that of the Form Definition Language Compiler; both command-line and forms-based interfaces will be provided.

3.2.3.4 SGML Context-Directed Editor (CDE)

The Context-Directed Editor is an EDS AP which guides a user through the process of document creation and revision by using the document type definition as a model for which logical elements may be included in a document. By informing the user of which logical elements are valid at any point within the authoring process, a document which conforms to a specified DTD is created.

The CDE differs from traditional word processors because it does not allow the layout style of the document to be specified when the document is formatted. The user can only "bind" text and attributes to logical elements within the document. The layout style for the document is defined by the Layout Macro Editor. As a result, the author of the document does not "see" the final form document as it is edited.

3.2.3.4.1 Inputs

The inputs to CDE are a document type definition, an existing document to be revised (if present), and user input of document content.

3.2.3.4.2 Outputs

The output of CDE is a document which contains both descriptive markup and document content.

3.2.3.4.3 Processing

When the user selects CDE from the EDS main menu, the form illustrated in Figure 3-17 is displayed.

```
+-----+
| IISS Electronic Documentation System      4/30/87 12:00 |
|                                           |
|           SGML Context-Directed Editor           |
|                                           |
| Document EDS.DS _____                |
|           DTD DSDOC _____              |
|                                           |
+-----+
```

Figure 3-17 CDE File Input Form

To complete this form, the user must enter the document to be edited and the name of the document type definition.

3.2.3.4.3.1 CDE Modes

When the user chooses either to edit an existing document or create a new document, the DTD for the document is read in and used as a template to guide the user through the process of creating a valid logical structure document. When a user edits a document, CDE will always be operating in two modes - text/edit and context (logical structure) mode. In the Text Edit mode, CDE uses the Form Processor Text Editing facility to perform standard text operations (ex. cut, paste, delete line, etc.) operations on the text of the document. In context mode, CDE uses the Application keypad (Figure 3-18) to define functions which operate on the logical structure of a document. All functions available via the Application keypad are described in the sections that follow.

Display Entity	Display Tags	Display Attribute	Display Context
Delete Logical Element			

Figure 3-18 CDE Application Keypad

3.2.3.4.3.2 CDE Document Editing

If the document specified to be edited exists, it is read in and displayed on the Document Editing form (Figure 3-19).

SECNO	SECTION 1
TITLE	SCOPE
HDR	1.1 Identification
P	This specification establishes the development, t
HDR	1.2 Functional Summary
P	EDS is
Context : <FM.SECTION> Document: EDS.DS Keypad Help = <KEY>	

Figure 3-19 CDE Document Editing Screen

The SGML tag names associated with the logical element text are displayed on the left-hand side of the form. This enables the user to know which text is being associated with which logical element.

The document name and the current context are displayed on the bottom of the form. The current context is an aggregation of tag names for the current logical element depth. As the user moves the cursor through a document, the context is updated. For example, when the cursor is positioned at the SECNO, the context would be <FM.SECTION.SECNO> indicating that the cursor is at a section number within a section within the front matter part of the document.

As the context changes, CDE lists only the SGML tags for logical elements which are valid in the current context. Valid tags can be viewed by pressing the <DISPLAY TAGS> key on the Application keypad.

SGML CDE

Valid TAGS for context: <SECTION>

<SECNO>
<TITLE>
<HDR>
<P>

Figure 3-20 Display Tags Form

As an example, if the current context is SECTION, the set of valid tags are any number of paragraph headers (HDR) or paragraphs <P>, but only one section number (SECNO) or title (TITLE). If the user presses the <DISPLAY TAGS> key, the form illustrated in Figure 3-21 will indicate the set of logical elements (TAGS) which are valid in the current context (SECTION).

When the user enters content for the SECNO and TITLE, the context changes and only HDR and P tag names are shown on the Display Tags form.

The editing process consists of a user supplying text for logical elements within a document. Text can be manipulated via commands in the Text Edit mode. As described below, the user can add logical elements to a context as long as they are valid. The context of the document can be changed at any time, in one of two ways:

- o <LIST CONTEXT> provides a list of all major logical elements in a document. To change context, select the context from the list and press <ENTER>. The document is positioned to the desired context.
- o SCROLL/PAGE mode allows the user to move through a document, changing context as the cursor moves.

The SGML CDE operates differently from typical word processors in that no formatting can be applied to text items in the document. Only the text for a given logical element can be entered. All word wrapping, lines between paragraphs, etc. are ignored until the document is processed by the Formatter.

3.2.3.4.3.3 Logical Element Editing

To add any valid logical element, position the cursor at the starting point for the logical element within the logical element form, then type in the name of the desired logical element name. CDE checks to see that the logical element is in a valid position.

To delete a logical element from a document, position the cursor to the logical element name and press <DELETE LOGICAL ELEMENT>. If the CDE determines that the logical element name is not required, then the logical element along with logical elements below the logical element content are deleted. For example, if a SECTION is deleted, all logical elements within that section are also deleted.

3.2.3.4.3.4 Document Creation

When a document which does not exist is specified, CDE generates the logical structure template for the first logical element of the document. The user can then proceed to supply the text for all logical elements on the form if text is required.

3.2.3.4.3.5 Optional and Required Logical Elements

The document type definition provides information to the CDE about which logical elements are required and which are optional within a document. All required elements are displayed in bold on the logical element form. The user must supply text for all logical elements which are required. In addition, no required logical elements can be deleted from the logical element form.

3.2.3.4.3.6 Using Entity Names in Documents

Entities in SGML enable the user to use abbreviations for long string names within a document. The string is then expanded as the document is processed by the PARSE. However, only Entity names which have been declared in the document type definition can be used. The <DISPLAY ENTITY> function displays the list (Figure 3-21) of all Entity names which have been declared in the DTD.

SGML CDF		4/30/87 12:00
Entity Names for DTD REQDOC		
&SGML	Standard Generalized Markup Language	
&CGM	Computer Graphics Metafile	

Figure 3-21 Display Entity Form

The user references an Entity by placing a & sign before the Entity name. For example, &SGML would be entered as opposed to the string Standard Generalized Markup Language.

3.2.3.4.3.7 Assigning Attributes to Logical Elements

Attributes are used in SGML to further qualify logical elements. The CDE enables the user to specify attributes for specific logical elements with the <DISPLAY ATTRIBUTE> function. This function displays all attributes and values for the current logical element. The user can then edit any attribute value for this logical element. Figure 3-22 illustrates the Display Attribute Form.

SGML CDE		4/30/87 12:00:00	
Attributes for Logical Element:			
<SECTION>			
Attribute Name			
security		Set	
id		u c ts ns	
		Value	
		u	

Figure 3-22 Display Attribute Form

3.2.3.5 SGML Layout Macro Builder

The Layout Macro Builder is a forms-driven EDS application program which assists in the generation of macro files which contain formatting instructions for classes of documents. These formatting instructions are used by the Formatter to direct the presentation of the text on an output device.

All macro files are associated with a unique document profile name. The document profile name is used to distinguish which macro files (and formatting instructions) are to be applied to an instance of a document. There may be multiple document profiles which could be used to format a particular document class.

The document profile contains a list of pointers from the generic identifiers in the DTD that define the logical view of the document elements to macro files that define the layout view of a document. The generic identifiers are expanded to allow the user to specify formatting instructions for specific parts of a document. For example, the user might wish to format a paragraph <P> in the preface <PR> differently than a paragraph in a section <SECT>. The Layout Macro program expands the generic identifiers to <PR.P> and <SECT.P> and allow the user to specify macro definitions for each one. When the document is imaged, paragraphs in the preface will have a different format than paragraphs in sections.

The layout macro files contain generic formatting instructions for each generic identifier found in the descriptive markup of a document. The user constructs macro files by interacting with a forms-based application that provides a menu of all possible formatting options that can be used to lay out a particular section of a document. To construct a macro for an expanded generic identifier, the user chooses which formatting options to use in the macro from the menu and supplies required values for the formatting options.

3.2.3.5.1 Inputs

The inputs to the Layout Macro application consist of user input via form fields of generic identifier to layout macro file references, document profile names, layout macro names, and selection of formatting options and values.

3.2.3.5.2 Outputs

The outputs of the Layout Macro application are document profiles and layout macro files. The document profile contains information about what document class the profile is to be used for, and a list of generic identifier to layout macro file name references. The layout macro file contains formatting language statements which define how the content, delimited by a generic logical identifier, will be presented on the output device.

3.2.3.5.3 Processing

3.2.3.5.3.1 DTD Selection Form

When the user selects the Layout Macro Editor from EDS Main Menu, a form (Figure 3-23) listing all existing Document Type Definitions is displayed.

```
+-----+
|                                     |
|               EDS  Layout  Editor   4/30/87 12:00:00 |
|                                     |
| Enter DTD Name or select from list > _____ |
|                                     |
| DTD Name           Description      |
| REQDOC             ICAM Requirements Document |
| DSDOC              ICAM Development Specification |
| IRSDOC             Internal Revenue Service Manuals |
| MEMODOC            General Purpose Memo |
| PSDOC              Program Specification |
|                                     |
+-----+
```

Figure 3-23 Layout Editor - DTD Selection Screen

This form enables the user to specify which document type to build layout macro instructions for.

3.2.3.5.3.2 Document Profile Form

Once a DTD is selected, the user must specify the document profile name. The document profile is a file which contains all macro files used for formatting a document. The Layout Macro program presents a form (Figure 3-24) which displays a list of all Document Profiles.

EDS Layout Macro Editor		4/30/87 12:00:00
Document Profile _____		___ Action (C, D, E)
Document Profile	Description	
___ REQDRFT	ICAM Requirements Document - DRAFT	
___ REQFNL	ICAM Requirements Document - FINAL	

Figure 3-24 Document Profile Form

Using this form, a user can EDIT, COPY, or DELETE existing document profiles or CREATE a new document profile.

3.2.3.5.3.3 Macro Selection Form

Once a user selects which document profile to edit, the Macro Selection form (Figure 3-25) is displayed.

EDS Layout Editor - Macro Selection 4/30/87	
DTD : REQDOC	Document Profile: REQDRFT
Generic Identifier	Macro Name
<SECTION.TITLE>	\$SECTTL
<SECTION.HDR1>	\$SECHDR1

Figure 3-25 Macro Selection Form

This form lists all expanded generic identifiers and the corresponding macro name, if one has been declared. In addition to the expanded generic identifiers, there is always a macro name of \$DEFAULT which contains all default formatting commands for the document.

The expanded generic identifiers are generated by reading the SGML elements in the document type definition and building a tree from the parent and child elements. A binding between formatting instructions and logical elements can only occur for those logical elements which are end nodes in the tree. For example, the following segment of a DTD can be also represented by tree structure shown in Figure 3-26.

```

<!ELEMENT  section  - - (sectno, title, (p | figure))>
<!ELEMENT  sectno   - - #PCDATA
<!ELEMENT  title    - - #PCDATA
<!ELEMENT  p        - - #PCDATA
<!ELEMENT  figure   - - (figbody, figcap)>
<!ELEMENT  figbody  - - NDATA
<!ELEMENT  figcap   - - (figno, figtxt) >
<!ELEMENT  figno    - - #PCDATA
<!ELEMENT  figtxt   - - #PCDATA

```

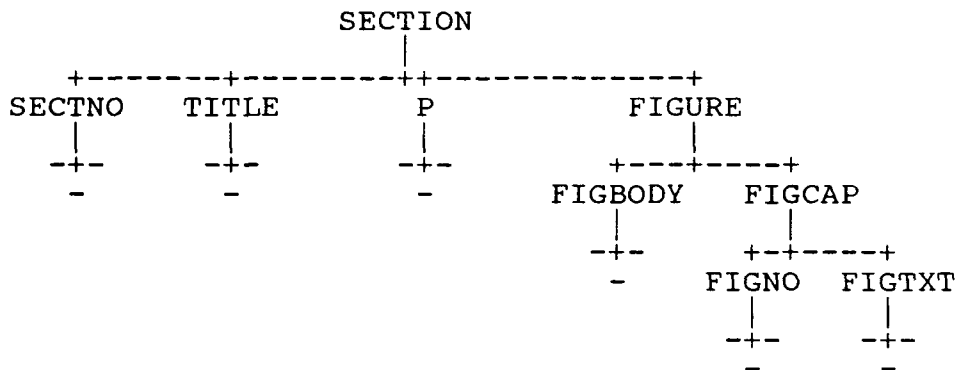


Figure 3-26 Tree Representation of DTD Segment

To generate expanded generic identifiers, a depth first search can be used. When the end of a tree segment has been reached, the expanded GI can be constructed from the names of all the nodes encountered along the way. Terminal nodes in the tree are typically those elements in the DTD which have #PCDATA in the logical element content. When a terminal node is encountered, the program must backtrack to a previous level and continue traversing other subtrees in the tree. The result of this traversal for the above example would be the following generic identifier names:

```

<SECTION.SECTNO>
<SECTION.TITLE>
<SECTION.P>
<SECTION.FIGURE>
<SECTION.FIGURE.FIGBODY>
<SECTION.FIGURE.FIGCAP.FIGNO>
<SECTION.FIGURE.FIGCAP.FIGTXT>

```

3.2.3.5.3.4 Format Selection

The user specifies the layout style of a logical element by selecting and supplying values for the formatting functions. The actual representation of the formatting functions will be numerous. The definition of formatting parameters for each previously defined document element will require several functions and separate forms for those definitions. An example of some formatting functions are shown in Figure 3-27.

DTD Name		:	REQDOC
Document Profile		:	REQDRFT
Macro Name		:	SECTTL

Bold	___	Font Size	___
Underline	___	Pitch	___
Reverse	___	Left-Indent	___
Italics	___	Right-Indent	___
		Center	___

Figure 3-27 Format Selection Form

The default value specified in the \$DEFAULT macro (or the system default value if none is specified in \$DEFAULT) is used for any formatting function which is not selected by the user but is necessary for formatting the document.

The three functions that are performed in this program are Page Setup, Generic ID Layout, and Headers/Footers. Each of these functions define specific layout or format requirements. Page Setup defines margins, line spacing, font style, font size, etc. The Generic ID Layout defines any format parameters that are specific to a document element. The Headers/Footers function defines the header information and the footer information for processing in the document.

When the user presses <ENTER>, signaling completion of the form, the Layout Editor program writes all formatting functions and values input by the user to the document profile macro.

3.2.3.6 SGML Formatter

The function of the Formatter is to create an output document which contains all control information necessary either to print or to display the document on an output device. In order to do this, the Formatter must map the layout style information contained in a document profile to the logical elements of a document. In addition, graphics files which are called for within the document are brought in for processing.

3.2.3.6.1 Inputs

The inputs to the Formatter are as follows:

- o User input of the document profile name to be used to process the document.
- o User input of the name of the document to be processed.
- o Layout macro files which are referenced within the document files.
- o External graphics files which are found in Entity references within the document content.

3.2.3.6.2 Outputs

The output of the Formatter is a document which consists of the document content and all Virtual Terminal Commands necessary to render the document on any output device supported by the Virtual Terminal. If errors are detected during the formatting process, an error report is also generated. These errors include unresolved entity names and generic identifier to layout macro file references.

3.2.3.6.3 Processing

When the Formatting option is selected from the EDS Main Menu, the form illustrated in Figure 3-28 is displayed.

IISS Electronic Documentation System 4/30/87 12:00	
Formatter	
Document Name	: EDS.DS
Document Profile	: DEVDRFT
Output File	: EDS.FMT
Print Document	: Y
Device Name	: SDRC\$PRINT
Device Type	: LN03

Figure 3-28 Formatter Input Form

This form enables the user to supply information about which document is to be formatted and which document profile is to be used to format the document.

The Formatter uses a three pass approach to process a document. The three passes are:

- o Pass 1 - Generic identifier (logical element) expansion/graphics file inclusion
- o Pass 2 - Logical element/layout style substitution

In Pass 1, the Formatter must expand the generic identifiers found in the document, match the expanded generic identifiers with a layout macro defined in the document profile for the class of document being processed, and merge all external graphics files referenced within the document. Expanded generic identifiers can be generated by using a stack-based approach when encountering SGML Tags. For example, a marked document might have the following appearance:

```
<FM>
<SECTION>
<SECTNO>SECTION 4</SECTNO>
<TITLE>QUALITY ASSURANCE</TITLE>
<P>This section</p>
</SECTION>
```

As the document is processed by the Formatter, generic identifiers are put on a context stack until the end tags are encountered. When document content is encountered, the expanded tag name is generated from the stack. This process then transforms the above document into this form:

```
<FM.SECTION.SECTNO>SECTION 4  
<FM.SECTION.TITLE>QUALITY ASSURANCE  
<FM.SECTION.P>This section
```

In pass 2, the Formatter matches the expanded generic identifier names generated by pass 1 with generic identifier names contained in the document profile. When a match is made, the Formatter reads the Layout Macro file which specifies what formatting attributes are applied to this logical element. Note that the Formatter at all times maintains a default attribute state which can be used in the event that there is no macro specified for a logical element, or an attribute is left unspecified in the layout macro (such as font or justification). The Formatter then inserts a neutral formatting language statement into the document which replaces the expanded generic identifier. Our sample document now looks like the following:

```
<PIR center, font_size = 14>  
SECTION 4  
<PIR center, underline, font_size = 10>  
QUALITY ASSURANCE  
<PIR left_indent=5, font_size = 8>
```

The use of a neutral formatting language to specify the layout style of logical elements is important because it does not preclude the use of multiple pass 3 translators. In addition to using the extended Virtual Terminal command set to image a document, the neutral formatting language commands can be translated into a page description language, or into a command set used by a typesetter. This flexibility enables EDS to support multiple output devices.

Pass 3 of the Formatter, as mentioned above, translates formatting commands embedded in the document into either the external Virtual Terminal protocol (which supports graphics) or into a formatting command set which can be processed by the desired output device.

3.2.3.7 Graphics File Translation

EDS includes application programs which enable translation between external vendor format to Computer Graphics Metafile (CGM). CGM is used within EDS as the standard format for the representation and storage of graphical data. These translators are user selectable via the Translate function listed on the EDS Main Menu.

3.2.3.8 Text File Translation

Text file translation from word processing formats to SGML is done by the SGML Tagger application program.

3.3 Performance Requirements

All EDS application programs are programmed using structured design and coding techniques. Basic programming standards for readability and ease of debugging are followed. EDS applications are implemented using the C programming language to insure the portability of EDS with a minimum of effort.

3.3.1 Program Organization

The modular design and the use of standards for document exchange and graphics files enable new applications to be added to EDS in a straightforward way.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process and correction of the cause of the error.

"Antibugging" is defined as the philosophy of writing programs in such a way as to make bugs less likely to occur and, when they do occur, to make them more noticeable to the programmer and to the user. This is done by incorporating as much error checking as possible in each routine.

4.2 Computer Programming Test and Evaluation

EDS as a document processing system will be tested by processing IISS documents which are currently in the WPS format. The documents will be processed by each EDS application program with the objective that the final form document produced by the Formatter should match the layout style of the original WPS document.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software is the Integrated Information Support System (IISS) Test Bed site located at Arizona State University, Tempe, Arizona. The software associated with each EDS CPCI release will be delivered on a medium which is compatible with the IISS Testbed. The release will be clearly identified and will include instructions to be followed for installation of the release.